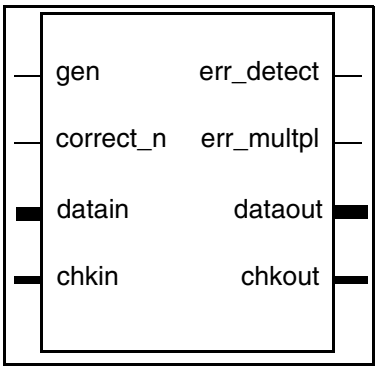
DW_ecc

Error Checking and Correction

Version, STAR and Download Information: IP Directory

Features and Benefits

Revision History



- Parameterized word width
- Generates check bits for new data written, and corrects corrupt data for read and read-modify-write cycles
- Supports scrubbing
- Flags to indicate if an error was detected, and if the error is not correctable
- Flow-through architecture for speed and flexibility
- Error syndrome output for error logging

Applications

- Highly-available and fault-tolerant mini-computers, personal computers, workstations, and mainframes
- Robust data buffering for data and telecommunications
- Error correcting and checking in peripherals, such as devices with embedded buffers
- Error correcting of external DRAM or SRAM buffers

Description

DW_ecc implements a modified Hamming code form of single-error correction, double-error detection (SECDED) error correction code (ECC) over a parameterized range of word widths.

DW_ecc uses redundant bits (check bits) in each word of storage (for example, in RAM). The check bits are encoded and written when the data bits are written. When the word is read, the check bits are used to determine whether one or more bits are in error and, if the error is a single-bit error, which bit contains the error.

DW_ecc guarantees detection of all one or two bit errors (including check bits). DW_ecc also guarantees correction of all single-bit errors (including check bit errors). DW_ecc does not guarantee error detection if more than two bits of a word (including check bits) are in error. In fact, if more than two bits of a word are in error, the error syndrome may match the error syndrome of a single-bit error, causing DW_ecc to miscorrect a single-bit error that does not exist.

For more details about the check bit equation, error syndrome map, and error flag behavior of DW_ecc, see the following SolvNetPlus article:

https://solvnetplus.synopsys.com/s/article/DW-ecc-error-syndrome-value-overview

Table 1-1 Pin Description

Pin Name	Width	Direction	Function
gen	1 bit	Input	Suppresses correction in write mode (`gen` = 1) and generates check bits Enables correction when in read mode (`gen` = 0) and `correct_n` is asserted (low).
correct_n	1 bit	Input	Enables correction of correctable words, active low
datain	*width* bits	Input	Input data word to check (check mode), or data from which check bits are generated (generate mode)
chkin	*chkbits* bits	Input	Check bits input for error analysis on read
err_detect	1 bit	Output	Indicates that an error has been detected, active high The error location is specified by the error syndrome.
err_multpl	1 bit	Output	Indicates that the error detected is uncorrectable Note that this pin does not always set to 1 when there are multiple bits in error. For a definition of uncorrectable errors for DW_ecc and the behavior of the `err_multipl` flag, see the SolvNetPlus article mentioned above.
dataout	*width* bits	Output	Output data May be corrected if an error is detected and `correct_n` is asserted.
chkout	*chkbits* bits	Output	■ When `gen` = 1, `chkout` contains the check bits generated from `datain` ■ When `gen` = 0 and *synd_sel* = 0, `chkout` is the corrected or uncorrected data from `chkin` ■ When `gen` = 0 and *synd_sel* = 1, `chkout` is the error syndrome value.

Table 1-2 Parameter Description

Parameter	Values	Description
width	4 to 8178	Width of input and output data buses The value of *width* impacts the minimum number of check bits that are required; for more, see Table 1-6 on page 4.
chkbits	5 to 14	Width of check bits input and output buses Calculate this value from *width* as listed in Table 1-6 on page 4.
synd_sel	0 or 1	Selects function of `chkout` when gen = 0 ■ 0: When `gen` = 0, `chkout` is the corrected or uncorrected data from `chkin` ■ 1: When `gen` = 0, `chkout` is the error syndrome value

Table 1-3 Synthesis Implementations

Implementation Name	Function	License Feature Required
str[a]	Synthesis model	DesignWare
rtl[b]	Synthesis model	DesignWare

a. For Design Compiler versions prior to 2009.06-SP1, the “str” implementation is used when *chkbits* is set to a value of '10' or less; “str” is not available for 2009.06-SP1 or later versions.
b. As of Design Compiler version 2009.06-SP1, “rtl” is the only synthesis implementation available and it supports all word sizes. Prior to 2009.06-SP1, the “rtl” implementation is used when *chkbits* is set to 11 or greater.

Table 1-4 Simulation Model

Model	Function
DW04.DW_ECC_CFG_SIM	Design unit name for VHDL simulation
dw/dw04/src/DW_ecc_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW_ecc.v	Verilog simulation model source code

Table 1-5 Error Checking and Correcting Truth Table

gen	correct_n	Mode	err_detect	err_multpl	Error Status / Action
1	0	Write; correction disabled	0	0	Detection and correction are disabled
1	1	Write; correction disabled	0	0	Detection and correction are disabled
0	0	Read; correction enabled	0	0	No error
0	0	Read; correction enabled	1	0	Error detected, corrected
0	0	Read; correction enabled	1	1	Multiple errors detected, data not modified
0	1	Read; correction disabled	0	0	No error
0	1	Read; correction disabled	1	0	Single-bit error detected, data not modified
0	1	Read; correction disabled	1	1	Multiple-bit error detected, data not modified

The number of check bits generated depends on the width of the data word. Table 1-6 lists the minimum number of check bits generated for a given value of *width*. The number of check bits is specified by the *chkbits* parameter.

Attention If you set *chkbits to* a value less than what is listed in Table 1-6, synthesis fails to build this component properly.

Table 1-6 Minimum Number of Check Bits Required

width	chkbits
4 to 11	5
12 to 26	6
27 to 57	7
58 to 120	8
121 to 247	9
248 to 502	10
503 to 1013	11
1014 to 2036	12
2037 to 4083	13
4084 to 8178	14

Figure 1-1 DW_ecc Block Diagram for *synd_sel* = 1

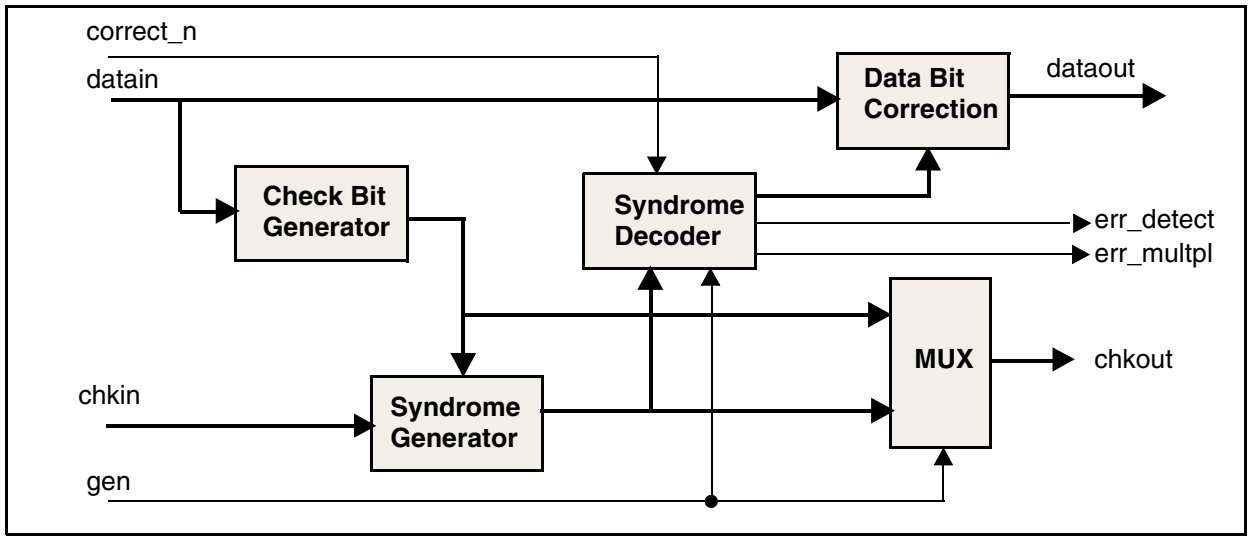
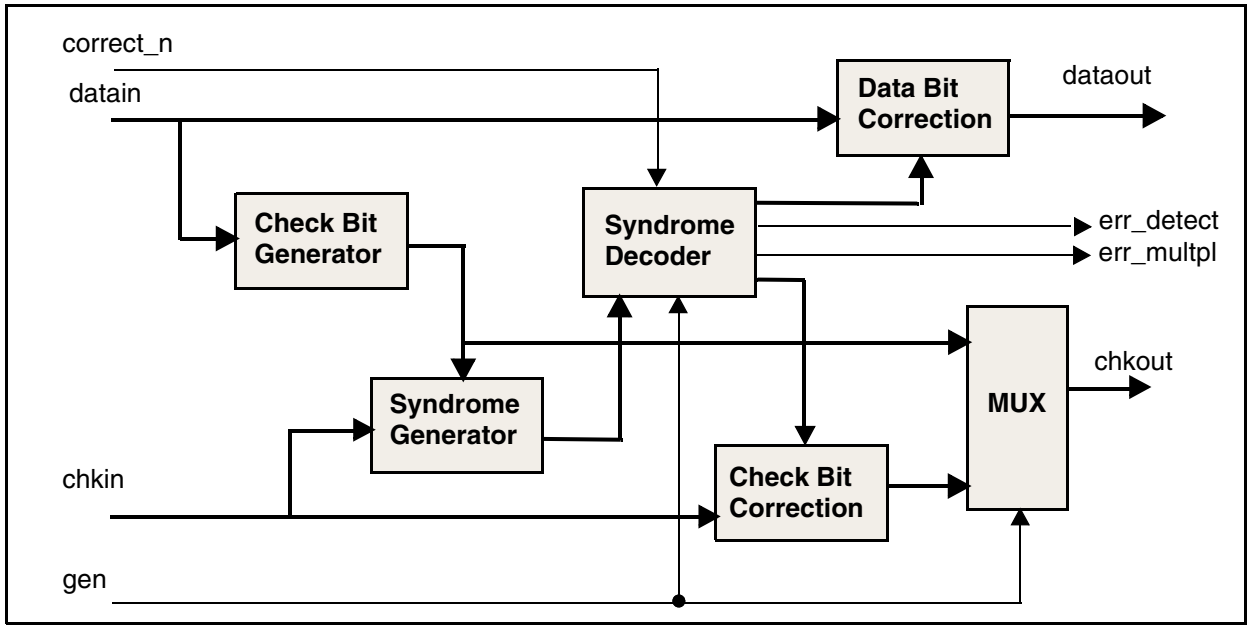


Figure 1-2 DW_ecc Block Diagram for *synd_sel* = 0



Writing to Memory

To generate check bits to be written to memory, set the input `gen` = 1 and apply the data (for which check bits are to be generated) to the `datain` inputs. This permits the data to propagate to the `dataout` outputs unchanged while the appropriate check bits are generated and output on the `chkout` port. Because DW_ecc is in the generate mode (`gen` = 1), the error flag outputs are disabled (`err_detect` = 0 and `err_multpl` = 0).

Reading from Memory

When the signal `gen` = 0 (indicating a read operation), the check bits are computed from the `datain` inputs, and XORed with the `chkin` inputs to form the error syndrome. If all bits of the syndrome are 0, then there is no error (recomputed check bits are the same as `chkin` inputs). If one or more syndrome bits are 1, then an error is detected as indicated by a the output `err_detect` = 1. If the error is not correctable, then the output `err_multpl` = 1 also. If parameter *synd_sel* = 1 and the input signal `gen`=0, then the error syndrome appears at the `chkout` output. The syndrome indicates which bit is in error, or whether multiple bits are in error, and is useful in systems where it is desirable to keep a log of which bits are chronically in error. Figure 1-1 on page 4 is a block diagram of DW_ecc when *synd_sel* = 1.

When correction is enabled (`correct_n` = 0 and `gen` = 0) and a correctable error is detected (indicated by `err_detect` = 1 and `err_multpl` = 0), the data bit in error is corrected as it passes from `datain` to `dataout`. If the error is in the check bits and parameter *synd_sel*=0, then the check bit in error is corrected as it passes from `chkin` to `chkout`. If the error is not correctable (indicated by `err_multpl` = 1), data passes unchanged from `datain` to `dataout`, and (if *synd_sel* = 0) check bits pass unchanged from `chkin` to `chkout`. Figure 1-2 on page 5 is a block diagram of DW_ecc when parameter *synd_sel* = 0.

When correction is disabled for a read operation, (`correct_n` = 1 and `gen` = 0), data passes unchanged from `datain` to `dataout` and, if *synd_sel* = 0, check bits pass unchanged from `chkin` to `chkout`. However, the status outputs, `err_detect` and `err_multpl` properly indicate the presence and the correctability of errors.

This mode of operation may be useful to control the use of correction; for example, in the case where data is to be corrected only when a bus fault is detected by the CPU.

DW_ecc implements single-error correction and double error detection (SECDEC). Therefore, when a two-bit error is encountered, both `err_detect` and `err_multpl` are high. Single-bit errors, however, are correctable, as indicated by `err_detect` high and `err_multpl` low. These conditions are illustrated in Table 1-5 on page 3.

Initializing Test Bench Memories With ECC Check Bits

The Verilog function, DWF_ecc_gen_chkbits, is provided in the file, $SYNOPSYS/dw/sim_ver/DW_ecc_function.inc. This function can be used in initial blocks to calculate check bits for the initialization of memories before simulation begins. To allow the function to be used with multiple configurations of DW_ecc, it accepts the data width and check bit width values as arguments along with the data to calculate check bits on and always returns a 16-bit result. For example if the three check bits are to be calculated on an 8-bit data value in the variable my_data, the function call might look like this:

```
`include "DW_ecc_function.inc"
  initial begin : initialize_stuff
    reg [15:0] dummy_chkbits;
    dummy_chkbits = DWF_ecc_gen_chkbits(8, 5, my_data);
    my_check_bits  = dummy_chkbits[4:0];
  end // initialize_stuff
```

This assumes that the variable my_check_bits is declared as a “reg” variable of size five bits and that the directory $SYNOPSYS/dw/sim_ver has been added to the simulator's include search path.

The DWF_ecc_gen_chkbits function is configured, by default, to support data widths up to 2048 bits. This can be changed by defining the Verilog macro, DW_ecc_func_max_width, up to the maximum allowed by the DW_ecc component.

The DWF_ecc_gen_chkbits function is provided for simulation only and is not capable of synthesis.

The DW_ecc_function.inc file contains a comment header that has additional information including the function argument descriptions. For the contents of this header, see “Header Information for DW_ecc_function.inc” on page 12.

Application Notes

In Example 1-1 and Example 1-2, three data bits are protected by four check bits. Although this example may seem rather simple, designing an EDC system can easily get rather confusing and unwieldy as word widths reach that of modern systems. In addition, there is a subtle method that can improve the likelihood of catching a certain percentage of triple bit errors, thus increasing system reliability even further. You can use the DW_ecc module to accomplish this.

The DW_ecc module's pass-through design fits easily into any system architecture. Most systems contain one instance of DW_ecc for check bit generation in the memory write path and another instance of DW_ecc in the memory read path for actual error detection and correction. For designers who want to also perform error scrubbing, the DW_ecc module can correct check bits as well as data bits. Some types of systems may take advantage of the ability (through parameterization) of the DW_ecc module to emit the error syndrome associated with an error. Designers of such systems may want to log the existence and location of errors that occur during operation. This information can be helpful in monitoring the "health" of the system, as well as aiding in locating faulty hardware.

Example 1-1 Encoding Equations

```
Check bit 0: C0 = D0 XOR D1
Check bit 1: C1 = D0 XOR D2
Check bit 2: C2 = D0 XOR D1 XOR D2
Check bit 3: C3 = D1 XOR D2
Encoding 110 results in a combined (data + check) of 0011.110
```

Example 1-2 Decoding Equations

```
Syndrome bit 0: S0 = D0 XOR D1 XOR C0
Syndrome bit 1: S1 = D0 XOR D2 XOR C1
Syndrome bit 2: S2 = D0 XOR D1 XOR D2 XOR C2
Syndrome bit 3: S3 = D1 XOR D2 XOR C3
Decoding 0011.110 results in an error syndrome of 0000 (i.e., no error)
Decoding 0011.100 results in an error syndrome of 1101, which uniquely identifies data
bit 1.
```

Figure 1-3 Memory Write Diagram

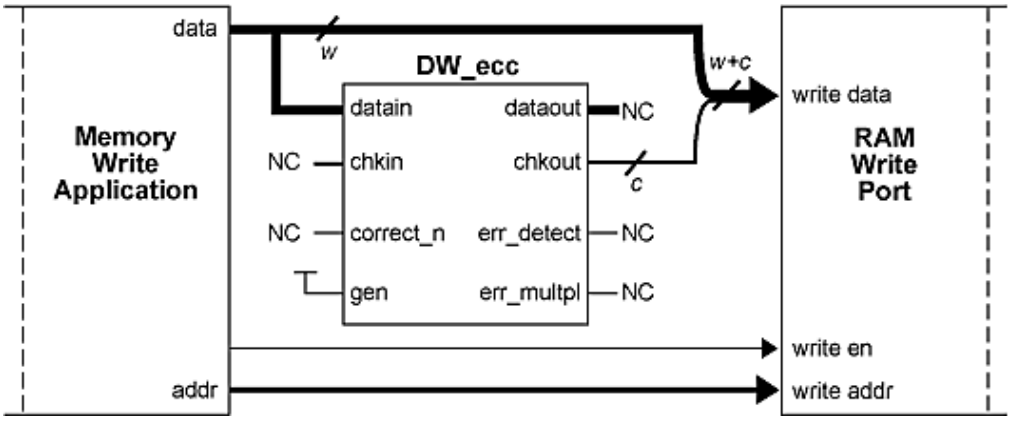
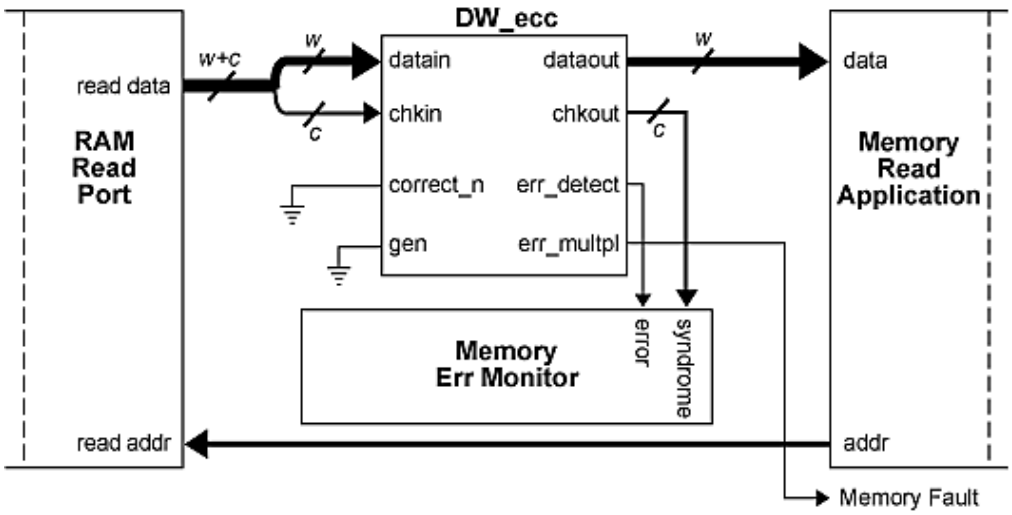


Figure 1-4 Memory Read Diagram with Syndrome Logging



Note For related usage and application information, see the following article:
"Generating checkbit equations and error syndrome map for DW_ecc"

Related Topics

- Application Specific – Data Integrity Overview
- DesignWare Building Block IP User Guide

HDL Usage Through Component Instantiation - VHDL

```
library ieee, DWARE;
use ieee.std_logic_1164.all;
use DWARE.DW_foundation_comp.all;

entity DW_ecc_inst is
  generic (wdth: integer := 64;
           cbts: integer := 8 );
  port (allow_correct_n   : in  std_logic;
        wr_data_sys       : in  std_logic_vector(wdth-1 downto 0);
        rd_data_mem       : in  std_logic_vector(wdth-1 downto 0);
        rd_checkbits      : in  std_logic_vector(cbts-1 downto 0);
        rd_error          : out std_logic;
        real_bad_rd_error : out std_logic;
        wr_checkbits      : out std_logic_vector(cbts-1 downto 0);
        rd_data_sys       : out std_logic_vector(wdth-1 downto 0);
        error_syndrome    : out std_logic_vector(cbts-1 downto 0) );
end DW_ecc_inst;

architecture inst of DW_ecc_inst is
  signal logic_0, logic_1 : std_logic;
  signal bus_of_0s : std_logic_vector(cbts-1 downto 0);
begin

  -- instance of dw_ecc for memory reads, uses rd_data_mem
  -- and rd_checkbits to derive corrected data on rd_data_sys,
  -- error flags on rd_error & real_bad_rd_error, and error
  -- syndrome value on error_syndrome

  U1 : dw_ecc
    generic map (wdth, cbts, 1 )
    port map (logic_0,            -- gen tied low for read
              allow_correct_n,    -- correct_n from system
              rd_data_mem,        -- datain from memory
              rd_checkbits,       -- chkin also from memory
              rd_error,           -- error flag to system
              real_bad_rd_error,  -- error flag for multi errors
              rd_data_sys,        -- read data (corrected if allowed)
              error_syndrome );   -- error syndrome for logging

  logic_0 <= '0';

  -- instance of dw_ecc for memory writes, uses wr_data_sys to
  -- generate wr_checkbits; output ports err_detect, err_multpl, and
  -- dataout are not connected

  U2 : dw_ecc
    generic map ( width => wdth, chkbits => cbts, synd_sel => 0 )
```

```
    port map (gen => logic_1,              -- gen tied high for write
              correct_n => logic_1,        -- correct_n not needed (tied high)
              datain => wr_data_sys,       -- datain from system
              chkin => bus_of_0s,          -- chkin not needed (tied low)
              chkout => wr_checkbits );    -- chkout written to memory
  logic_1 <= '1';
  bus_of_0s <= (others => '0');
end inst;
```

HDL Usage Through Component Instantiation - Verilog

```
module DW_ecc_inst(allow_correct_n, wr_data_sys, rd_data_mem,
                   rd_checkbits, rd_error, real_bad_rd_error,
                   wr_checkbits, rd_data_sys, error_syndrome);
  parameter wdth = 64;    // 64-bit data busses
  parameter cbts = 8;     // 8-bit check field

  input allow_correct_n;
  input [wdth-1:0] wr_data_sys, rd_data_mem;
  input [cbts-1:0] rd_checkbits;
  output rd_error, real_bad_rd_error;
  output [wdth-1:0] rd_data_sys;
  output [cbts-1:0] wr_checkbits, error_syndrome;

  // instance of DW_ecc for memory reads, uses rd_data_mem
  // and rd_checkbits to derive corrected data on rd_data_sys,
  // error flags on rd_error & real_bad_rd_error, and error
  // syndrome value on error_syndrome

  DW_ecc #(wdth,cbts,1)
    U1 (.gen( 1'b0 ),                    // gen tied low for read
        .correct_n( allow_correct_n ),   // correct_n from system
        .datain( rd_data_mem ),          // datain from memory
        .chkin( rd_checkbits ),          // chkin also from memory
        .err_detect( rd_error ),         // error flag to system
        .err_multpl( real_bad_rd_error ), // severe error flag to system
        .dataout( rd_data_sys ),         // read data (corrected if allowed)
        .chkout( error_syndrome ) );     // error syndrome for logging

  // instance of DW_ecc fro memory writes, uses wr_data_sys to
  // generate wr_checkbits

  DW_ecc #(wdth,cbts,0)
    U2(1'b1,                  // gen tied high for write
       1'b1,                  // correct_n not needed (tied high)
       wr_data_sys,           // datain from system bus
       { cbts{1'b0} },        // chkin bus not needed (tied low)
       ,                      // no error flag for writes
       ,                      // no multi error flag for writes
       ,                      // no data modification for writes
       wr_checkbits );        // check bits written to memory
endmodule
```

Header Information for DW_ecc_function.inc

```
////////////////////////////////////////////////////////////////////////////////
//
// NOTE: THIS FUNCTION IS NOT FOR SYNTHESIS. It is provided for use in
//       initializing ECC check bits for memories at time zero of simu-
//       lations (such as in an initial block with 'for' loops).
//
//
// Function: DWF_ecc_gen_chkbits
//
//     Calculates the check bits to be used with the data passed (based on
//     the data width and check bit width arguments passed) that are directly
//     compatible with the logic of the DW_ecc design with parameter values
//     that match the same data and check bit width values.
//
// Arguments:
//     width : number of data bits to calculate check bits from (integer)
//     chkbits : number of check bits to be generated (integer)
//     data_in : data bits to be used to calculate check bits from (vector
//               of size `DW_ecc_func_max_width)
//
// Returns:
//     16-bit value with the specified number of check bits in the low
//     order bits.
//
//
// The Verilog macro, DW_ecc_func_max_width should be set to a value that
// is equal to or greater than the largest value of the width argument passed
// to DWF_ecc_gen_chkbits.  Its default setting (if not defined before the
// DWF_ecc_function.inc file is included) is 2048 but the DW_ecc design can
// support widths up to 8178.  This macro can also be used in the design
// that includes it to size the vector that is passed as the data_in argument
// of the function call (to avoid lint issues).
//
// The DW_ecc_gen_chkbits function always returns a 16-bit value, no matter
// how many check bits are specified with the chkbits argument.
//
////////////////////////////////////////////////////////////////////////////////
```

Revision History

For notes about this release, see the *DesignWare Building Block IP Release Notes*.

For lists of both known and fixed issues for this component, refer to the STAR report.

For a version of this datasheet with visible change bars, click here.

Date	Release	Updates
October 2022	DWBB_202202.5	■ Corrected capitalization of Hamming on page 1
June 2022	DWBB_202203.2	■ Clarified descriptions of error detection and correction behavior and added a link to a SolvNetPlus article for more details in "Description" on page 1 ■ Clarified the function of `err_multipl` in Table 1-1 on page 2
September 2020	DWBB_202009.0	■ Moved description of the relationship between *width* and *chkbits* to precede Table 1-6 on page 4
March 2019	DWBB_201903.0	■ Added "Application Notes" on page 7 ■ Added this Revision History table and the document links on this page

Copyright Notice and Proprietary Information

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://www.synopsys.com/company/legal/trademarks-brands.html.

All other product or company names may be trademarks of their respective owners.

Free and Open-Source Software Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
www.synopsys.com